UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Sistema de Predicción para Carreras de Fondo

Gonzalo Muelas Pozo
Tutor: Gonzalo Martínez Muñoz

Julio 2017

Sistema de Predicción para Carreras de Fondo

AUTOR: Gonzalo Muelas Pozo
TUTOR: Gonzalo Martínez Muñoz

Dpto. Computación e Ingeniería del Software
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2017

Resumen

Este Trabajo Fin de Grado ha consistido en la realización de un estudio sobre la predicción de tiempos en carreras de fondo. El objetivo es la predicción del resultado que se obtendría en una prueba de distancia maratón, basándose en datos reales de corredores. Para ello, en lugar de utilizar métodos tradicionales de predicción de marcas (como VO2MAX o VAM), se ha hecho uso de algoritmos de aprendizaje automático que ofrecen una mejor aproximación al resultado real, ya que tienen en cuenta un mayor número de factores y variables.

Para ello se dispone de una base de datos que contiene información sobre entrenamientos y carreras reales de corredores anónimos, así como datos sobre su complexión física, su edad, el desnivel del recorrido, la distancia total de la carrera, la distancia de los intervalos del recorrido, datos de tiempos en los entrenamientos, etc. Debido a que estos datos son reales, fiables y contrastados, los algoritmos de aprendizaje automático deberían predecir de una forma más exacta los resultados de los usuarios en la carrera objetivo.

Antes de poder usar los datos de los que disponemos, ha sido necesaria una transformación de los mismos a un formato con el cual pudiéramos trabajar. Primero se han realizado varios *scripts* para procesar los ficheros de datos (en formato CSV) y poblar la base de datos (en SQLite3). Los algoritmos utilizados se han implementado en Python, debido a la existencia de librerías de aprendizaje automático de gran calidad, como *scikit learn*.

Se han tenido en cuenta diversos algoritmos y se ha realizado un estudio sobre la eficacia de los mismos, aportando datos estadísticos y representaciones gráficas de los resultados.

Adicionalmente, con el objetivo de ofrecer un servicio de predicción para corredores interesados en mejorar sus marcas y utilizar esta información como referencia en sus entrenamientos, se ha implementado un servicio web basado en una arquitectura *RESTful*. El servidor se ha construido sobre un *framework* en Python, *Django*. Antes de realizar esta elección, se ha realizado un pequeño estudio previo sobre qué plataforma utilizar para la implementación del servicio. La API REST, que se ha implementado y probado posteriormente a través de *scripts* y pruebas manuales, nos permite interactuar con la base de datos de forma fácil a través de la web, así como generar las predicciones a partir de un modelo de aprendizaje.

Palabras clave

Maratón, Predicción, Python, Django, Scikit Learn, REST, API, Aprendizaje Automático, Inteligencia Artificial, Aprendizaje Supervisado, Regresión, Web Framework.

Abstract

This Bachelor Thesis aims to perform a study about the efficiency of automatic learning algorithms prediction in long-distance races. The goal is to predict the result of real runners in future long-distance races, using real data in a database. Automatic learning algorithms will substitute traditional methods to make time predictions (such as VO2MAX or VAM), as they approximate better to real times, because they make use of a greater number of variables and parameters.

To do so, we have at our disposal a database that records information about trainings and goal races of real anonymous runners, as well as some additional data such as physical build, age, slope of the route, total race length, length of intervals, time marks when training, etc. Due to the reliability of the data, algorithms should predict more efficiently the results of the runners in a goal race.

Before using data it has been necessary to transform them to a format we could manage. By using some python scripts, we have processed the files (in csv format) and then populated the database (SQLite3). Also automated learning algorithms have been implemented in python, using *scikit learn* library, which is very accurate for our purpose.

Several algorithms have been considered and we have studied their efficiency, attaching data and graphical representations of the results.

Additionally, to let the interested users be able to enhance their marks using our service, we have implemented a web server based on a *RESTful* architecture using the Python *framework* Django. Before choosing this one we have compared some of them to see which one fits better. REST API allows us to interact easily with the system through the web, as well as generate a prediction from a trained learning model.

Keywords

Marathon, Prediction, Python, Django, Scikit Learn, REST, API, Automatic Learning, Artificial Intelligence, Supervised Learning, Regression, Web Framework.

Agradecimientos

Para empezar, agradezco a mi tutor por guiarme en este proyecto, en especial sobre cómo tratar los datos y los algoritmos de aprendizaje automático. También a la empresa RUNATOR por compartir con nosotros sus datos, que son la parte más importante de este trabajo, en mi opinión. A mi familia y amigos por darme ánimos para sacar adelante el trabajo. También agradezco estos cuatro años en la Universidad Autónoma, donde me he formado como ingeniero informático y donde he conseguido mi primer empleo gracias al programa de prácticas externas.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1 Introducción

Este proyecto ha sido realizado con esfuerzo y con gran interés. Se engloba dentro del campo de la inteligencia artificial aunque depende fuertemente de una base importante relacionada con los sistemas informáticos, sin la cual sería imposible de realizar. A continuación se describen los aspectos que lo han motivado y los objetivos que se han planteado realizar.

1.1 Motivación

Este proyecto se encuentra motivado en otras cosas por el creciente interés por el *running* en nuestra sociedad. Cada vez más personas deciden dedicar tiempo a este deporte como forma de hacer ejercicio diario, sin necesidad de gastar apenas dinero en su realización, ya que se realiza en plena calle y con un gasto en material pequeño en relación con otros deportes. Los datos muestran un gran auge de este deporte [1], que se ha convertido en un fenómeno mundial.

Además se han comercializado en los últimos años multitud de accesorios, como *smartwatches* o *pulseras de actividad*, que miden nuestro ritmo cardíaco, detectan nuestros pasos, el desnivel, la distancia recorrida, disponen de acelerómetro, giroscopio, brújula, pulsómetro, barómetro, altímetro, geolocalizador (GPS), etc [2].

Todas estas funciones han motivado un aumento del número de aplicaciones que se desarrollan en función de estos dispositivos. Estas aplicaciones permiten controlar el ejercicio diario y ayudan de forma considerable a los corredores habituales. De esta forma resulta mucho más fiable obtener los datos estadísticos que necesitamos para llevar a cabo este proyecto. Este trabajo esta dedicado de alguna forma a esas personas que realmente dedican tiempo a este deporte y que entrenan de forma constante para obtener una buena marca en carreras de maratón. Sus entrenamientos son la base de esta investigación, y se han utilizado para crear este sistema de predicción.

El trabajo llevado a cabo en este TFG, consiste en la realización de un sistema basado en servicios web capaz de predecir un tiempo para una carrera de maratón basándose en una serie de entrenamientos realizados por el corredor en cuestión obtenidos de la red social deportiva Runator.
De cuantos más entrenamientos se disponga, mayor será la precisión de la respuesta.
Por otro lado, se ha desarrollado una API web sencilla intentado que el servicio sea accesible vía web, de forma que sea accesible por los corredores y por otras aplicaciones como Runator.

1.2 Objetivos

El objetivo de este proyecto es poder ayudar a corredores de maratón interesados en conocer su evolución, aportándoles una estimación temporal de la marca que obtendrían en una carrera real basándose en sus entrenamientos pasados. El proyecto pretende ser un punto de inicio para que la empresa que nos facilitó los datos iniciales pueda incorporar este nuevo servicio en una posible herramienta.

Para ello, es necesario primero analizar el problema y decidir qué tipo de algoritmos y teoremas utilizar, y conocer hasta qué punto es posible realizar el proyecto. Una vez que sabemos que es posible, necesitamos datos reales y fiables a partir de los cuales entrenar nuestro modelo de aprendizaje.

Una vez obtengamos los datos, es necesaria una fase de tratamiento y procesamiento de los mismos, para eliminar anomalías y datos erróneos o que rompan las estadísticas de forma que puedan empeorar los resultados de la estimación.

Los datos preparados deben incluirse en una base de datos, para que el acceso a éstos sea más rápido y eficaz. Además necesitamos poner en marcha el servidor con el cuál nos vamos a comunicar. Debemos elegir una arquitectura y un *framework* adecuados.

El siguiente objetivo es preparar los datos para que los algoritmos de aprendizaje automático los puedan comprender y puedan trabajar con ellos.

A continuación deberemos probar diferentes algoritmos para comprobar cuál de ellos funciona mejor y de qué forma podemos minimizar el error. Esta fase es la más compleja ya que debemos realizar pruebas sobre varios parámetros de configuración de los algoritmos y extraer nuestras conclusiones, aportando pruebas que sustenten nuestra elección.

Por último debemos incorporar nuestro modelo de aprendizaje dentro del servidor, con el objetivo de ofrecer servicio web que permita a corredores reales obtener una predicción en una futura carrera.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Introducción: se ha realizado una breve descripción del proyecto, resumiendo los temas más importantes a tratar. Es importante conocer los objetivos a desarrollar y qué ha motivado este trabajo. En este apartado se ha dado a conocer la magnitud del proyecto, qué es lo que se quiere conseguir.

- Estado del arte: en todo proyecto es muy conveniente conocer cuál es la base teórica en que éste se ha basado para situarnos desde un punto de vista crítico y poder formar nuestra opinión al respecto del tema tratado. Se ha descrito el

estado actual de los diversos campos tratados, investigando previamente qué estudios, publicaciones, corrientes, etc existen al respecto.

- Diseño: Explicación de la estructura del proyecto y de las decisiones de diseño que han sido tomadas en el transcurso del trabajo de fin de grado. Se ha ilustrado mediante diagramas y figuras para que resulte más fácil de comprender el proyecto en su totalidad.

- Desarrollo: Una vez esté claro el boceto del proyecto, qué se va a realizar, la idea general que abarca y qué objetivos que se plantean, es momento de explicar en detalle cómo se han llevado a cabo dichos objetivos a un nivel más técnico.

- Integración, pruebas y resultados: Una vez hemos desarrollado nuestros subsistemas, es necesario realizar pruebas sobre ellos para minimizar y controlar los errores que puedan existir, y decidir si el proyecto va por buen camino. En el caso de software, podemos realizar casos de uso y realizar pruebas sobre los módulos implementados. Cuando éstos pasen la pruebas de forma satisfactoria, podemos realizar la integración y realizar a su vez pruebas sobre el sistema final.

- Conclusiones y trabajo futuro: Es la fase de valoración de nuestro proyecto, donde decidimos qué objetivos hemos logrado cumplir y cuáles no, y hasta qué punto el trabajo tiene la funcionalidad deseada en un principio. Además es posible que el trabajo sirva como base de otro proyecto futuro de un estudiante, una empresa u otro organismo.

- Referencias: Lista de fuentes de información contrastadas que nos han sido útiles para desarrollar el proyecto (libros, revistas, estudios, publicaciones, artículos, *posts*, etc).

- Glosario: Recopilación de términos específicos relacionados con nuestro trabajo y que requieren una aclaración o descripción.

- Anexos: Son añadidos a nuestro trabajo que nos sirven para explicar algunos de los procesos realizados más en profundidad, y que no suponen una parte fundamental del proyecto aunque son aclaratorios.

2 Estado del arte

En esta sección vamos a conocer la base técnica de este proyecto, sobre qué teorías se sustenta y qué estudios se han llevado a cabo en este campo. Esto nos dará una base para entender las decisiones tomadas durante el desarrollo del trabajo.

Para ponernos en contexto, este trabajo consta de dos partes muy bien diferenciadas. Por un lado hemos creado un **servicio web** que nos permite ejecutar la aplicación desarrollada y dar acceso a otros programas a través de un API a las distintas funcionalidades.

Por otro lado, este proyecto se basa en un campo ampliamente estudiado y en cuyas bases teóricas nos hemos basado para desarrollarlo, la **inteligencia artificial** y el **aprendizaje automático**. Hemos utilizado algoritmos como regresión mediante árboles de decisión, regresión lineal o redes neuronales como el perceptrón multicapa.
A continuación vamos a describir ambos campos algo más en profundidad.

2.1 Arquitectura del servicio web

En esta sección, vamos a conocer por qué hemos elegido una arquitectura de tipo de *RESTful* [3]. Además vamos a realizar una pequeña comparativa entre algunos *frameworks* para desarrollo web, concretamente en el lenguaje Python.

Existen diversas arquitecturas de servidores. Entre ellas podemos destacar aquellas orientadas a servicios (SOA) basadas en RPC (Remote Procedure Call), en las cuales los usuarios de un servicio web transfieren sobre el protocolo HTTP a un servidor datos e información para ejecutar un procedimiento remoto. Un *servicio web* se basa en protocolos de redes de comunicación entre sistemas con distintas arquitecturas y lenguajes de programación, que les permite comunicarse entre ellos en el mismo formato (por ejemplo para ejecutar una rutina remota). **SOA** (*Arquitectura Orientada a Servicios*) es un paradigma de computación que utiliza un conjunto de servicios interactivos como forma de desarrollo de aplicaciones [4]. **RPC** (*Llamada a Procedimiento Remoto*) es una rutina que ejecuta código en otra máquina remotamente gracias a los protocolos de red que permiten el entendimiento entre ambos sistemas.

Los datos que se intercambian pueden estar representados en diferentes formatos y sobre diferentes arquitecturas lo que da lugar a variantes como SOAP, RMI, CORBA, etc.
Este tipo de arquitecturas no aprovechan todo el potencial del protocolo HTTP, ya que se ejecutan normalmente sobre el método POST y disponen de una única dirección de destino (*endpoint*) que representa el servidor destino. De esta forma, el servidor recibe una petición y ejecuta el método especificado con los argumentos requeridos que ha recibido.

Frente a esta forma de estructurar un servicio web, surge a principios de este siglo el concepto de arquitectura *RESFful*, término que se utiliza para describir aquellos servicios que utilizan una forma de representación **REST** (*Representational State Transfer*). Los servicios de este estilo buscan una forma ligera y rápida de intercambiar información con los clientes directamente en HTTP, aprovechando al máximo las características de este protocolo. Para ello, cada tipo de objeto o recurso del servidor va a corresponder a una

dirección URL, denominada URI (*Uniform Resource Identifier*). La forma de intercambiar los datos (referido a los argumentos de un método RPC, por ejemplo), es variada pero la más común es mediante representación en formato JSON (*Javascript Object Notation*), que se usa tanto para enviar información desde el cliente como para recibir información del servidor. Además REST utiliza todos los verbos disponibles en HTTP (GET, POST, PUT, DELETE, PATCH, OPTIONS, etc) como forma de implementar el paradigma de un sistema informático CRUD (*Create, Read, Update and Delete*). De este modo se establecen las acciones necesarias a realizar sobre cierto tipo de recurso u objeto de un sistema.

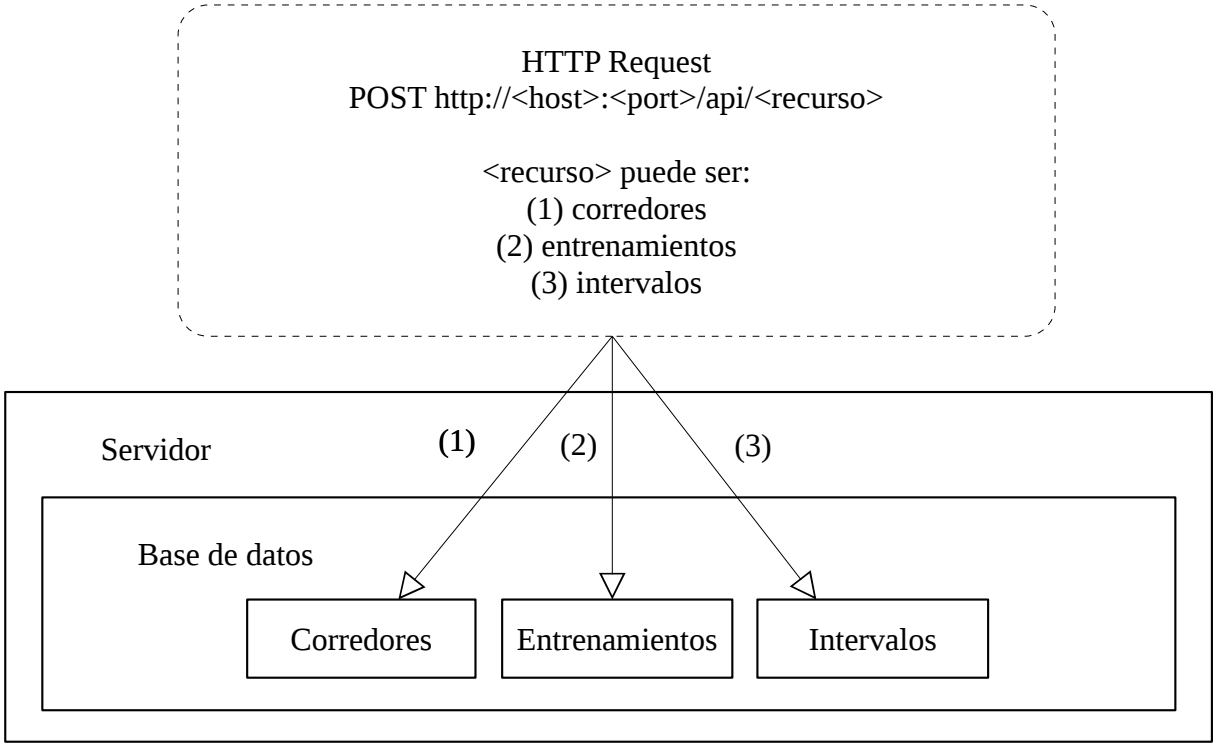


Figura Estado del arte-1: Arquitectura REST del servidor

En la Figura 1 (ejemplo de nuestro servicio web), añadimos un nuevo recurso a la base de datos mediante una petición HTTP con el método POST. También podemos leer recursos mediante el método GET o borrarlos con DELETE. La url de la petición es el identificador del objeto o URI. Además, ésta debe ir acompañada de una cabecera HTTP, en la cual se indicará el tipo de contenido (JSON) y la autenticación en el servidor, y un cuerpo, que contiene una estructura de tipo JSON con los datos del recurso que vamos a añadir. En general el comportamiento de los verbos no es fijo, sino que se puede definir en cada servicio, aunque conviene que sigan el siguiente estándar: GET para leer, POST para crear, PUT para actualizar y DELETE para borrar.

Hemos elegido esta arquitectura para nuestro servidor, considerando que es más apropiada para nuestro proyecto. Como ya se ha explicado anteriormente, uno de los objetivos de este proyecto es facilitar a la empresa que nos ha proporcionado los datos una forma de insertar corredores y entrenamientos en la base de datos para mejorar así el modelo de aprendizaje, aumentando el número de datos que se usan para entrenarlo. Bien, pues REST nos permite

insertar de una forma muy sencilla objetos en nuestra base de datos mediante peticiones HTTP. También hemos definido ciertas URL para generar un modelo entrenado, y para realizar finalmente la predicción de un tiempo de un determinado usuario en la carrera de maratón.

En este punto conviene explicar ciertas decisiones sobre la arquitectura del servidor. Para implementarlo se pensó inicialmente en un **lenguaje de programación** versátil que contase con *frameworks* para desarrollo web y librerías de aprendizaje automático. Finalmente, este último hecho fue decisivo para decidirse por Python.

Hemos utilizado la librería *scikit learn* ya que está enfocada a Python. Al ser una librería implementada en lenguaje compilado de bajo nivel como C, es bastante rápida y útil para nuestra necesidades, no queremos que tarde demasiado en entrenar el modelo de aprendizaje ya que recordemos que el tiempo es muy importante en los servicios web. Por otra parte queríamos que el modelo de aprendizaje estuviera integrado en el servidor, y por tanto el proyecto en su totalidad ha sido implementado en Python.

2.2 Framework

Como *framework* de desarrollo web hemos elegido **Django**, una plataforma de alto nivel, que automatiza ciertas funciones de un servidor, tanto *back end* como *front end*, haciendo mucho más sencilla su implementación. Tras realizar una comparativa entre varios sistemas alternativos al uso de Django (Flask, Pyramid, Tornado, Bottle, Diesel, Pecan, Falcon, etc), se ha llegado a la conclusión de que éste es uno de los más completos, ya que no requiere librerías externas, como en otros casos, e incorpora métodos de autenticación, *routing*, base de datos propia (SQLite3), plantillas y *templates* para la creación de formularios, panel de administración, etc. Otro factor clave para decidirnos, ha sido la gran comunidad de usuarios que Django posee, que desde su primera *release* en 2006 ha ido ampliando la documentación de la herramienta, así como el número de ejemplos de código. Uno de los módulos de Django llamado *django rest_framework*, el cual se encuentra ampliamente documentado en internet, ha sido fundamental en el desarrollo del servidor.

2.3 Predicciones en carreras de fondo

Existen una serie de metodologías clásicas para la predicción de tiempos en carreras de fondo y maratones. La mayoría de métodos se basan en fórmulas matemáticas, que dan un valor temporal a partir de los tiempos obtenidos en pruebas más cortas multiplicados por un coeficiente. Tradicionalmente se han venido utilizando métodos de predicción de marcas como VO2MAX o VAM. Por ejemplo, VO2MAX mide el máximo volumen de oxígeno que puede consumir un atleta en mililitros por kilogramo de peso corporal por minuto (ml/kg/min) [5] y a raíz de estos datos obtendríamos la marca predicha.

Pero estos métodos aunque no producen un fallo demasiado elevado, no permiten la flexibilidad que nos aportan los métodos basados en aprendizaje automático y requieren de la realización de una prueba por parte de los atletas.

Como referencia, podemos encontrar el anterior trabajo de fin de grado "*Predicción en carreras de fondo*" [6], el cual muestra cómo los métodos de aprendizaje automático se aproximan bastante bien a las marcas reales en las pruebas de maratón.

2.4 Aprendizaje Automático

Lo primero que debemos plantearnos es el tipo de problema que debemos resolver, y cómo podemos aplicar en él métodos de aprendizaje automático. Además, ¿por qué es interesante resolver este problema mediante aprendizaje automático?, ¿qué ventajas tiene? Primero vamos a conocer qué es y las principales aplicaciones de esta rama de la inteligencia artificial. Entre las definiciones más aceptadas encontramos la siguiente:

[7] *"El **aprendizaje automático** es el campo de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos. Es, por lo tanto, un proceso de **inducción del conocimiento**."*

El comportamiento de una máquina puede cambiar ante la entrada de nuevos datos. Permite obtener un enunciado general a partir de casos particulares. Sin embargo, es prácticamente imposible obtener una inducción completa del problema, ya que siempre existe una incertidumbre, ya sea porque no se han proporcionado suficientes ejemplos, éstos estén incompletos o no se hayan procesado los datos de una forma adecuada. Es por ello, que siempre existirá cierto grado de error en el resultado proporcionado por el algoritmo de aprendizaje automático.

Por un lado, podemos clasificar los métodos de aprendizaje en función de si entrenan conociendo la solución al problema o "a ciegas". De una forma más concreta, nos encontramos los siguientes grupos:

- **Aprendizaje supervisado**: el algoritmo conoce la solución (el objetivo o clase) de cada uno de los datos de entrenamiento, por tanto, va adaptándose a medida que avanza la fase de entrenamiento. Muchos problemas se pueden resolver de esta manera si se cuenta con un número de mediciones adecuado para que el algoritmo minimice el error de manera aceptable. El número de algoritmos que se basan en el aprendizaje supervisado es enorme, y entre ellos, podemos destacar los clasificadores y regresores basados en estadística bayesiana, vecinos próximos, regresión logística, redes neuronales como los perceptrones (con o sin retro propagación), árboles de decisión o conjuntos de clasificadores.

- **Aprendizaje no supervisado**: en este caso, los datos no van acompañados de los valores objetivo, por lo que no existe conocimiento del problema a priori. Se utiliza para la compresión de datos o para la agrupación de elementos (*clustering*) con una determinada distribución. El algoritmo o la red neuronal utilizada para resolver el problema, descubre características o patrones similares en los datos de entrada y es capaz de agruparlos en consecuencia. Algunos ejemplos son los algoritmos de c*lustering* o las *redes neuronales de Kohonen*.

Existe un gran número de problemas que se nos pueden presentar. Dentro del marco que nos interesa en este proyecto, que es el **aprendizaje supervisado,** podemos encontrarnos varios grupos según el **tipo** de objeto que intenten **predecir**.

- **Clasificación**: el objetivo es predecir una clase de entre un número predefinido de ellas. Puede ser clasificación binaria si sólo tenemos dos clases, o multiclase si nos encontramos varias clases. Este tipo de enfoque se puede aplicar por ejemplo para detectar si un paciente sufre una determinada enfermedad o no (clasificación binaria) o para detectar letras en escritura manual (clasificación multiclase).

- **Regresión**: los objetivos en estos casos son valores reales, es decir debemos predecir un valor continuo y no discreto. Se puede utilizar para predecir el valor de la bolsa en un determinado momento, para reproducir series temporales o para predecir la marca en segundos en una prueba de maratón por ejemplo.

- **Ranking**: trata de predecir el orden óptimo de un conjunto de elementos según su relevancia. Por ejemplo, es útil para realizar recomendaciones de música o películas a los usuarios de una plataforma.

A continuación vamos a describir los algoritmos que hemos utilizado en este trabajo desde un punto de vista técnico.

2.4.1 Algoritmos

Nuestro problema radica en predecir tiempos o marcas en carreras de larga distancia como maratones. Si analizamos los grandes grupos descritos con anterioridad, nos daremos cuenta de que debemos aplicar algoritmos de aprendizaje automático de **regresión** donde nuestras predicciones serán valores reales expresados en segundos.

Nuestros datos se componen de carreras reales realizadas por corredores anónimos, en las cuáles tenemos como objetivo (*target*) el tiempo que éstos consiguieron en la carrera. Por tanto podemos deducir también que se trata de aprendizaje **supervisado**.

Una vez conocemos estas características, debemos elegir qué algoritmos son más adecuados para el problema. En nuestro caso hemos considerado los siguientes algoritmos supervisados de regresión, esto es con objetivo continuo.

2.4.1.1 Regresión Lineal

[8] En términos estadísticos, la **regresión lineal** es un modelo matemático que se utiliza para establecer una relación de dependencia entre la variable dependiente Y, las variables independientes Xi y un término aleatorio ε.

$$Y_t = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$$

donde:

Y_t es la variable objetivo dependiente o variable a predecir,
X_i son las variables independientes o variables que se utilizan para la predicción,
β_i son los parámetros que miden la influencia sobre las variables independientes, β_o es el término constante y p es el número de dimensiones del problema,
ε es un término aleatorio, que recoge todos los casos no controlables u observables.

El caso más sencillo se da cuando tenemos un sólo parámetro independiente y por tanto el hiperplano obtenido como resultado de la transformación es una **recta**.
Cuando debemos estimar p parámetros β_i, la estimación de la variable dependiente viene dada por el **hiperplano** definido por la siguiente ecuación:

$$Y_i = \beta_0 + \sum \beta_{pi} X_{pi} + \varepsilon_i$$

El modelo cumple que $\varepsilon_i \sim N(0, \sigma^2)$, por lo que $E[\varepsilon_i] = 0$ y $Var(\varepsilon_i) = \sigma^2$ (condición de homocedasticidad). Existe normalidad en las perturbaciones aleatorias.

Las rectas de regresión son aquellas que se ajustan mejor a la nube de puntos como podemos ver en la figura 2..

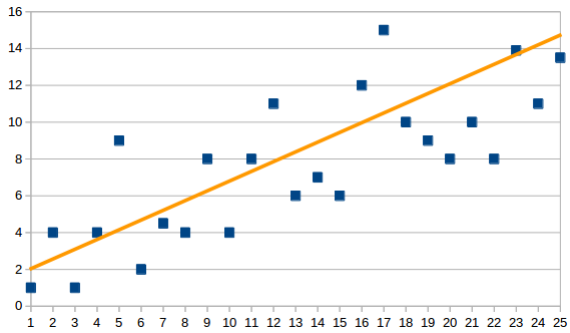


Figura Estado del arte-2: Recta de regresión lineal simple

2.4.1.2 Random Forest Regression

El *Bosque Aleatorio* o *Random Forest* tiene como base otros clasificadores más sencillos. Es un conjunto de árboles de decisión o *Decision Trees*, en los cuáles las entradas se van procesando a través del árbol hasta dar un valor de salida para la variable dependiente. El Random Forest para regresión promedia las salidas de estos árboles con el objetivo de dar una predicción más precisa.

Para entrenar cada uno de los T árboles del modelo *Random Forest* se hace lo siguiente:

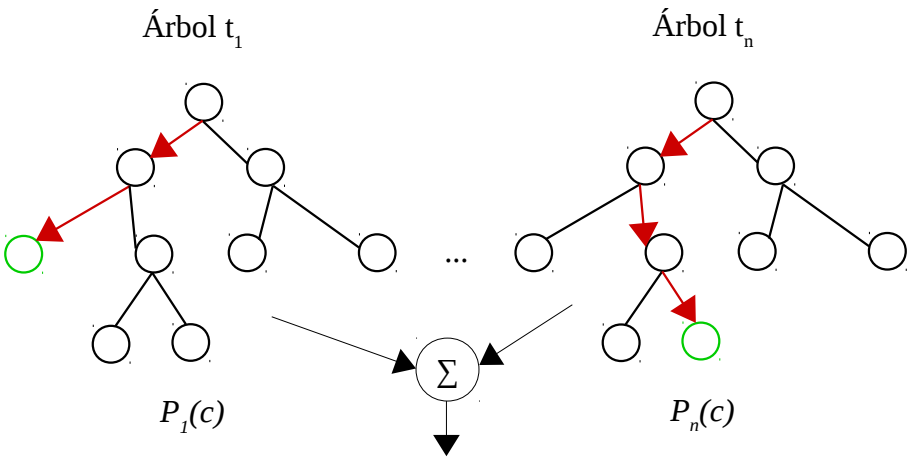
1. Creamos un dataset más reducido que el original, a partir de una muestra *bootstrap*. Esta muestra se genera a partir de nuestra muestra original efectuando un muestreo con reemplazamiento de los valores. Se efectúan un gran números de mediciones para estimar la distribución de los parámetros.

2. Para cada nodo del árbol:

    1. Se selecciona un número aleatorio de variables *m* tal que $m \leq p$, donde *p* es el número total de variables del problema.

    2. Se realiza una partición binaria en el nodo a partir de una función o condición favorable sobre las variables seleccionadas para el nodo. Los datos se dividen entre los que cumplen la condición y los que no, dando lugar así a dos nodos hijos.
    3. Se repite la operación anterior para los nodos hijos recursivamente, seleccionando de nuevo *m* variables de entre los parámetros posibles y creando nuevos nodos.

El número de variables *m* nos da diferentes tipos de árbol dependiendo de su valor. Los valores utilizados generalmente son *m*=1 (una única variable), *m=p (todas, equivalente al algoritmo bagging)*, $m=\sqrt{p}$, o *p* como estándar utilizado por *Random Forest*.



[9]

$$P(c|atrib)=\sum_{t=1}^{T} P_t(c|atrib)$$

Figura Estado del arte-3: Random Forest

Como vemos en la figura, el resultado del *Random Forest* está condicionado por la salida de otros árboles. El resultado puede ser una media aritmética o ponderada de todos los nodos terminales resultantes o un voto por mayoría si se trata de variables categóricas.

2.4.1.3 Perceptrón multicapa regresor

Una red neuronal artificial trata de imitar el comportamiento de las redes neuronales biológicas. Éstas están compuestas por neuronas, células nerviosas que "disparan" una señal eléctrica cuando reciben cierto estímulo a la entrada. A la función que regula dicha señal se le denomina *función de activación* o de *transferencia (f(x))* en una neurona artificial.

$$y_{\text{in}} = b + \sum_{i=1}^{p} x_i w_i$$

$$y = f(y_{\text{in}})$$

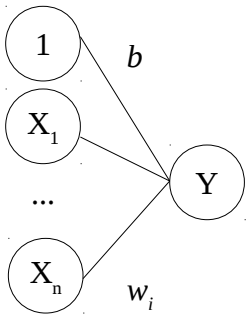


Figura Estado del arte-4: Esquema de un Perceptrón Simple

En la figura 4 se representa un *Perceptrón Simple*, una red de una sola capa, donde X_i son las entradas a la red, w_i son los pesos para las conexiones entre la entradas y la salida, *b* es el peso para la entrada llamada sesgo, e *Y* es el valor de salida que se obtiene mediante las fórmulas expuestas.

La salida de la red depende de la función de activación. Algunas de las funciones que se suelen utilizar son:

- Sigmoide Binaria: $f(x) = \dfrac{1}{1+\exp(-x)}$
- Sigmoide Bipolar: $f(x) = \dfrac{2}{1+\exp(-x)} - 1$
- Función escalón binaria: $f(x) = \begin{cases} 1 \, si \, x > 0 \\ 0 \, si \, x \geq 0 \end{cases}$
- Función escalón bipolar: $f(x) = \begin{cases} 1 \, si \, x > 0 \\ -1 \, si \, x \geq 0 \end{cases}$

El *Perceptrón Multicapa* es un tipo de red neuronal similar al *Perceptrón Simple*, pero con un mayor número de capas intermedias.

Muchos algoritmos se basan en configurar los pesos de las conexiones entre las neuronas para que la red "aprenda". Las entradas tomarán los valores de las variables del problema, y se propagarán hacia la capa siguiente multiplicándose por los pesos de las conexiones. Esta operación se repite hasta obtener un valor de salida.

Por cada ejemplo en nuestro problema se realizará una iteración, en la cuál los pesos de la red serán modificados. Estos cambios en los pesos pueden obtener una configuración adecuada para clasificar correctamente los datos estudiados [10].

En algunas variantes de este algoritmo se calcula el error obtenido en cada iteración (*backpropagation*), y se retropropaga hacia la capa de entrada. Luego, estos errores se tienen en cuenta para modificar los pesos de forma más precisa.

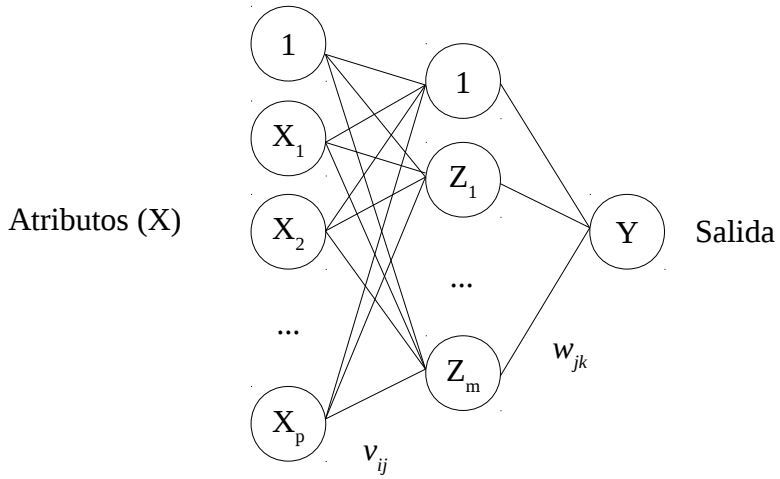


Figura Estado del arte-5: Estructura del Perceptrón Multicapa

En la figura 5 podemos ver la estructura de un *Perceptrón Multicapa* con una capa intermedia.

2.4.2 Scikit Learn

Existen varias librerías en Python para el desarrollo de aplicaciones de aprendizaje automático. En este proyecto, hemos trabajado con *scikit learn* [11], una librería implementada en C que es eficiente computacionalmente y que es perfecta para nuestras necesidades, ya que contiene multitud de algoritmos de distintos tipos, incluidos los descritos anteriormente.

Para trabajar con esta librería es necesario importarla a nuestro proyecto, así como otros módulos dentro de ésta.

```
import sklearn
from sklearn.linear_model import LinearRegression
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import RandomForestRegressor
```

Usar los regresores es sencillo, tan sólo es necesario instanciarlos, configurando los parámetros iniciales que se consideren necesarios, entrenarlos y predecir la salida.

Es conveniente realizar un entrenamiento y una validación con varias particiones, para obtener un resultado promedio más aproximado. En nuestro caso hemos utilizado el módulo *Kfold* que pertenece a *sklearn.cross_validation* para realizar **particiones cruzadas** de los datos, de forma que todos los ejemplos se usen una única vez en la fase de validación.

A continuación, en la Figura 6, mostramos un ejemplo de entrenamiento y predicción utilizando la librería ***scikit learn***.

```
X = datos[:,:-1]
X_normalized = preprocessing.normalize(X, norm='l2')
Y = datos[:,-1]

# Particiones con validación cruzada

part = KFold(len(datos), 10, shuffle = True)

# Recorremos las particiones

for train_index, test_index in part:

      # Extraemos los datos correspondientes a train y test

      datosTrain = X_normalized[train_index,:]
      datosTest = X_normalized[test_index,:]
      clasesTrain = Y[train_index]
      clasesTest = Y[test_index]

      # Entrenamos el modelo

      model.fit(datosTrain, clasesTrain)
      pred = model.predict(datosTest)
```

Figura Estado del arte-6: Ejemplo de entrenamiento y predicción de un modelo

Adicionalmente hemos utilizado la librería ***matplotlib*** [12] de Python para crear los recursos gráficos del proyecto. Nos ha sido útil para representar las marcas reales de las carreras de maratón frente a las predicciones obtenidas.

2.5 Redes sociales deportivas

Existen en el mercado una gran cantidad de aplicaciones móviles que permiten a los corredores llevar un seguimiento de sus entrenamientos y su progreso (Runkeeper, Endomondo, Runtastic, Nike+, Strava, Polar o Garmin). Sin embargo, estas aplicaciones tienen diferencias que dificultan la colaboración entre varios corredores.

2.5.1 Runator

Esta empresa es una *start up* de origen español que se dedica al desarrollo de servicios y aplicaciones informáticas en el sector del *running*.
[13] *Runator* nace con el objetivo de crear un perfil de corredor a partir de los resultados obtenidos en las aplicaciones de *running* que existen el mercado. De esta forma es posible visualizar el progreso de otras personas a través de su perfil de *Runator*, todo desde una misma aplicación. No se trata de una aplicación de *running*, sino que monitoriza la actividad del resto aplicaciones. Así puedes comparar tu progreso con los demás y ver tu clasificación respecto al resto de corredores.

Ya que *Runator* posee información sobre entrenamientos de corredores así como de sus marcas en carreras de maratón reales, ha sido posible disponer de estos datos para realizar el proyecto.

Para más información sobre *Runator*, visite https://www.runator.com/es/

3 Diseño

En esta sección seguiremos un orden lógico. Trataremos los temas según se han desarrollado en el proyecto. Comenzaremos con el tratamiento de los datos en los ficheros originales, seguiremos con el diseño de la base de datos y el servidor, y por último los algoritmos de aprendizaje automático.

3.1 Diseño de la Base de Datos

Inicialmente, nuestros datos se encuentran en ficheros en formato *csv* que debemos tratar para volcarlos a una base de datos. En la fase de desarrollo se explicará el procesado de los fichero *csv*, por ahora nos centraremos en el diseño de la base de datos.

Tenemos dos entidades fundamentales, los ***corredores*** y los ***entrenamientos***, aunque también poseemos información interna de éstos últimos, en forma de ***intervalos***. Es muy interesante el desglose de un entrenamiento en intervalos más pequeños ya que de esta forma conocemos más al detalle como se ha desarrollado el entrenamiento, y podemos enriquecer los atributos de entrada de los algoritmos.

Veamos estas **entidades** más en profundidad:

Un **corredor** se identifica de forma única mediante un *id*. Los atributos que siguen (se muestran en la Figura 7) corresponden a la carrera objetivo que se está tomando como referencia y se dispone de la siguiente información: distancia, fecha de la carrera, ciudad y país donde se celebró, tiempo obtenido por el corredor, género y edad.

Un **entrenamiento** está identificado de la misma forma por un *id*, pero a su vez está vinculado a un corredor determinado en la entidad previa mediante una clave foránea. Los atributos (mostrados en la figura 8) más destacables que determinan el entrenamiento son: la fecha en que se ha realizado, la distancia recorrida y el tiempo obtenido. Además, un entrenamiento se compone de intervalos más pequeños como hemos mencionado que son útiles para saber los ritmos internos dentro del entrenamiento.

Un **intervalo** también tiene su propio identificador, y éste está vinculado a un entrenamiento mediante una clave foránea. Otros atributos destacables (mostrados en la figura 9) son: la distancia del intervalo, el tiempo empleado para recorrer esa distancia, y el orden que ocupan en el entrenamiento al que pertenecen (cada entrenamiento tiene varios intervalos ordenados).

La base de datos se compone de las tablas mostradas en las figuras 7, 8 y 9. Como se puede observar un corredor tiene N entrenamientos que se muestran en la figura 8. Por ejemplo, el corredor con id 1, ha realizado dos entrenamientos, los cuales tienen identificadores 1 y 2. A su vez, el entrenamiento 1 está compuesto por los intervalos 1, 2 y 3, y el entrenamiento 2 por los intervalos 4 y 5 (como se aprecia en la figura 9).

Corredor										
id	distancia	fecha	ciudad	pais	desnivel_total	tiempo_media	tiempo_final	velocidad_media	genero	edad
1										
2										
3										
4										
5										
6										
7										

Figura Diseño-7: Tabla Corredor

Entrenamiento							
id	id_corr	app	fecha	distancia_total	tiempo_total	tiempo_media	desnivel_total
1	1						
2	1						
3	2						
4	3						
5	3						
6	4						
7	5						

Figura Diseño-8: Tabla Entrenamiento

Intervalo					
id	id_entr	distancia_intervalo	tiempo_intervalo	media_intervalo	orden
1	1				1
2	1				2
3	1				3
4	2				1
5	2				2
6	3				1
7	3				2

Figura Diseño-9: Tabla Intervalo

Las relaciones entre las tablas se pueden ver en el diagrama *Entidad-Relación* mostrado en la Figura 10.

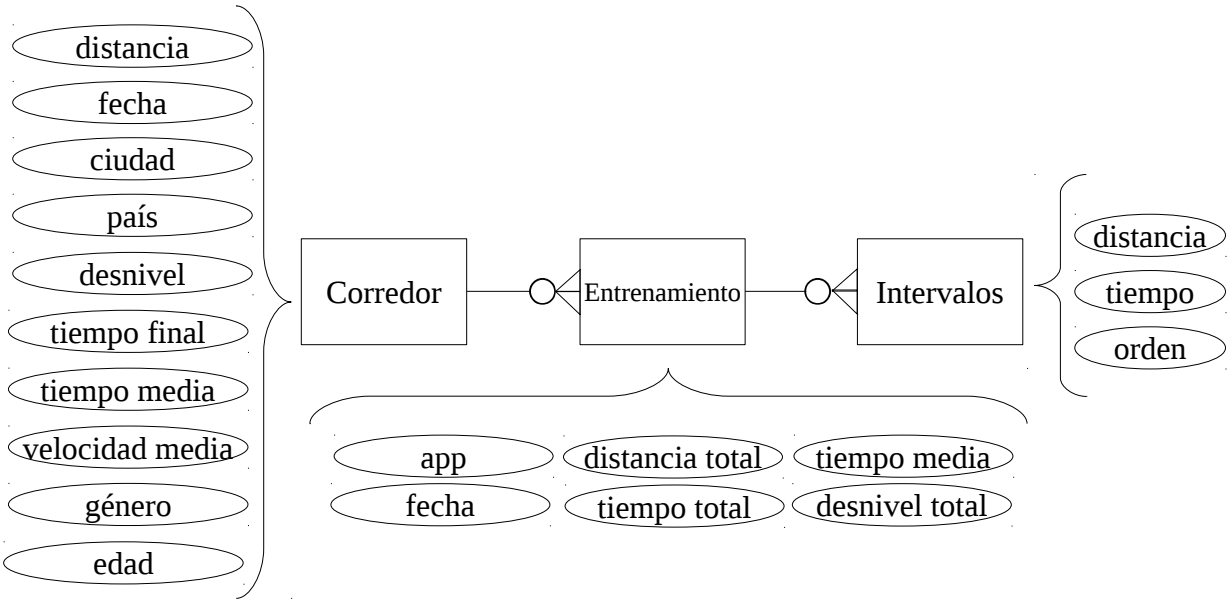


Figura Diseño-10: Diagrama Entidad-Relación

3.2 Servidor

Como ya se ha mencionado, el servidor sigue una arquitectura REST, de forma que todos los recursos del sistema se encuentran referenciados por una URI. Esta dirección nos permite acceder mediante la URL del navegador de forma única. Adicionalmente el modo en el que accede al recurso mediante un verbo del protocolo HTTP, determina el tipo de operación que se realiza sobre el recurso.

El sitio web se ha estructurado en torno a los siguientes recursos.

- **Página de inicio**: Muestra una página con una interfaz gráfica amigable, desde la cual puedes acceder a distintos puntos de la aplicación mediante un menú de navegación (ver figura 11). Además, la página de inicio muestra una guía de uso tanto de la API como del protocolo REST (ver figura 12), para que no haya dudas acerca del uso de la API REST y la autenticación en el servidor. Es posible acceder a la página de inicio desde la siguiente dirección de acceso:

  http://<host>:<port>/

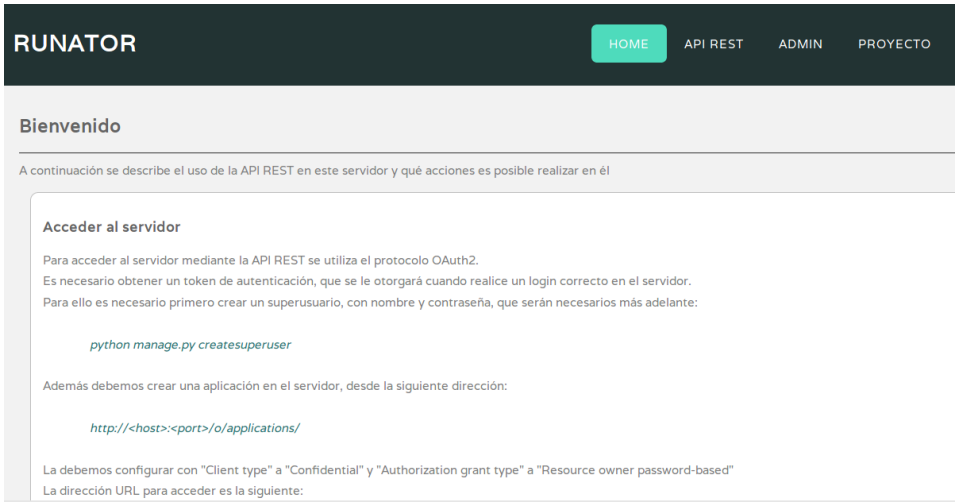


Figura Diseño-11: Página Principal

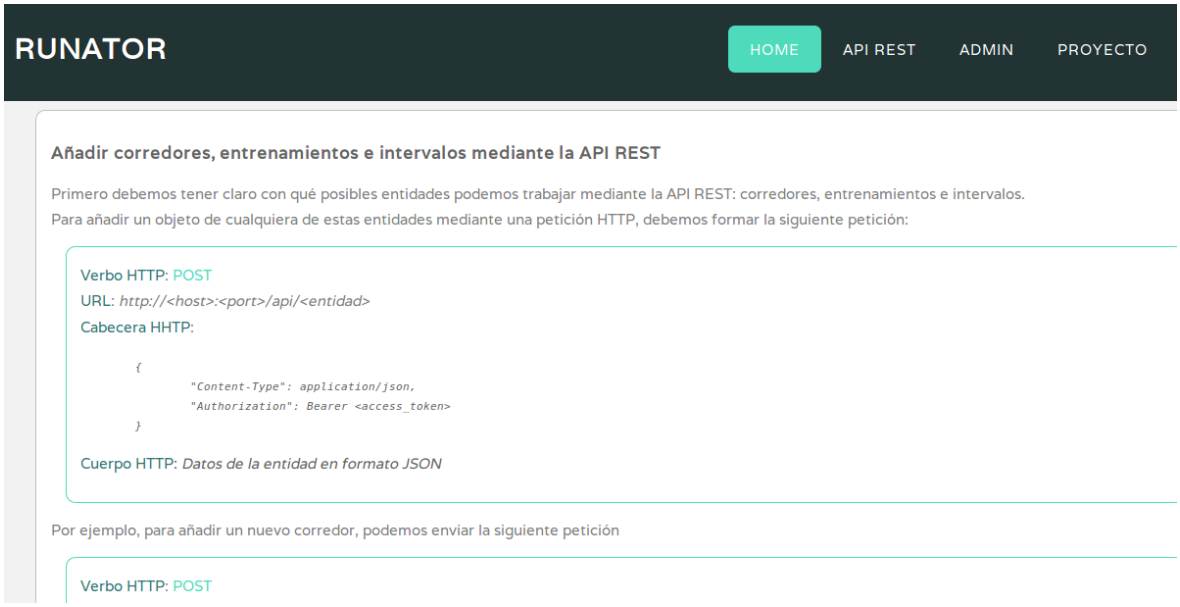


Figura Diseño-12: Guía sobre la API REST

- **Panel de administración de Django**: Para acceder a él necesitamos autenticarnos, habiendo previamente creado un superusuario en el sistema (Véase *Manual del Programador, Anexo B*). Desde aquí podemos administrar nuestras instancias de la base de datos, como los corredores, los entrenamientos y los intervalos, así como las aplicaciones creadas. Las aplicaciones son la forma de utilizar la API de una forma segura, ya que nos proporcionan unas credenciales de acceso, distintas para cada aplicación. Podemos acceder desde

  http://<host>:<port>/admin/

  donde se nos pedirá la autenticación. Podemos el panel de administración en la figura 13.

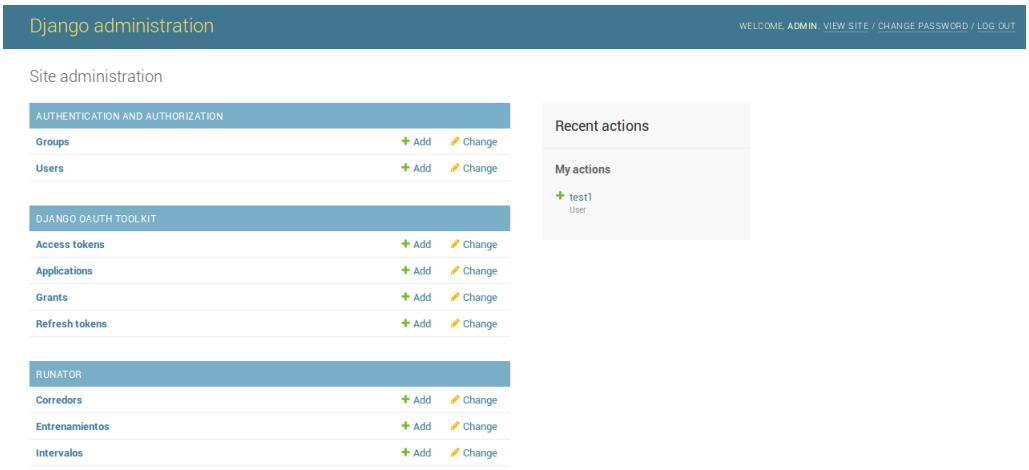


Figura Diseño-13: Panel de Administración de Django

- **Panel principal de la API:** En este panel se pueden ver las distintas entidades definidas, esto es, los corredores, entrenamientos e intervalos. Adicionalmente, desde este panel se puede interactuar con la API. Funciona de forma parecida a una API interactiva, con interfaz gráfica en lugar de comandos y peticiones HTTP en bruto. La URL raíz es http://<host>:<port>/api/ , seguida del nombre del recurso deseado, esto es:

  http://<host>:<port>/admin/corredores

  http://<host>:<port>/admin/entrenamientos

  http://<host>:<port>/admin/intervalos

  El la figura 14 se muestra el panel para la entidad de corredores.

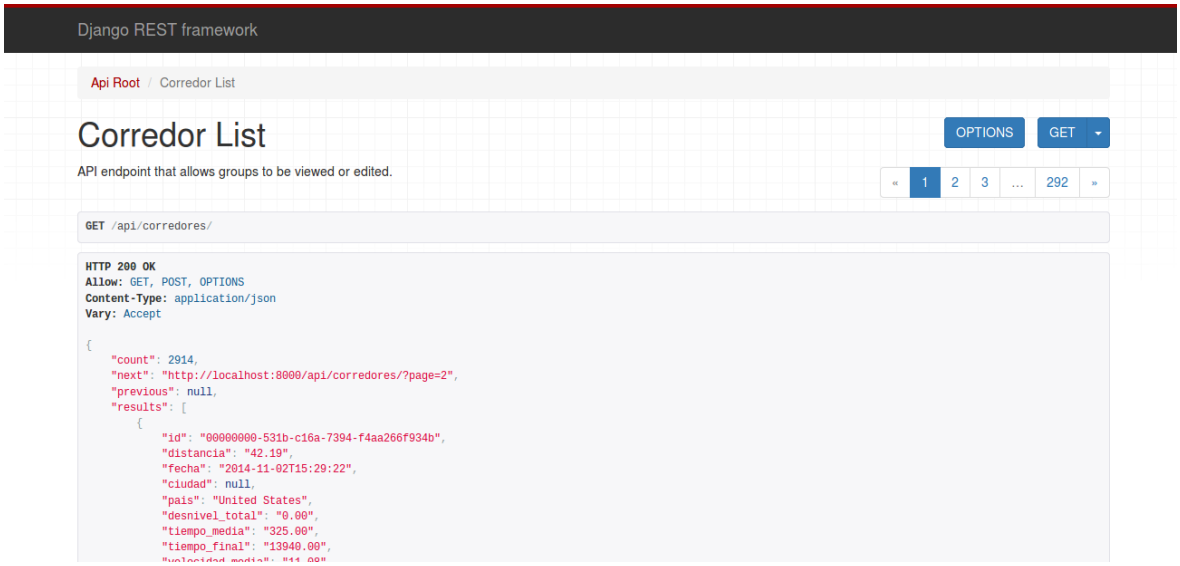


Figura Diseño-14: API REST Interactiva, Lista de corredores

- **Generar modelo**: No es viable entrenar nuestro modelo o algoritmo cada vez que recibamos una petición para predecir una marca. Por ello se ha decidido añadir una función que se encarga de entrenar el modelo y almacenarlo **serializado** en disco. Es suficiente con entrenarlo una vez, a no ser que se añadan más datos a la base de datos, en cuyo caso conviene actualizarlo para generar un modelo más preciso. De esta forma la fase de validación que nos dará el valor predicho será muy rápida. En la figura 15 se muestra la página de ayuda de esta función de la API. La URL de acceso es:

  http://<host>:<port>/generar_modelo/

Figura Diseño-15: Generar Modelo de Aprendizaje

- **Predecir una marca**: Para obtener una predicción sobre la marca de un determinado corredor debemos dirigirnos a la dirección

http://<host>:<port>/predecir/<id_corredor>

y obtendremos el tiempo que el algoritmo de aprendizaje automático predice para el corredor especificado.

Figura Diseño-16: Predecir una marca

3.3 API REST

Vamos a describir las principales funciones de la API REST. Contamos con tres tipos de entidades u objetos, ya descritos en el apartado 3.1 (Diseño de la Base de Datos).
Gracias a los métodos HTTP podemos realizar las siguientes operaciones.

3.3.1 Añadir objetos mediante la API REST

Para añadir *corredores*, *entrenamientos* e *intervalos* a la base de datos utilizamos peticiones POST con la siguiente estructura:

Figura Diseño-17: Estructura general de una petición para añadir un objeto

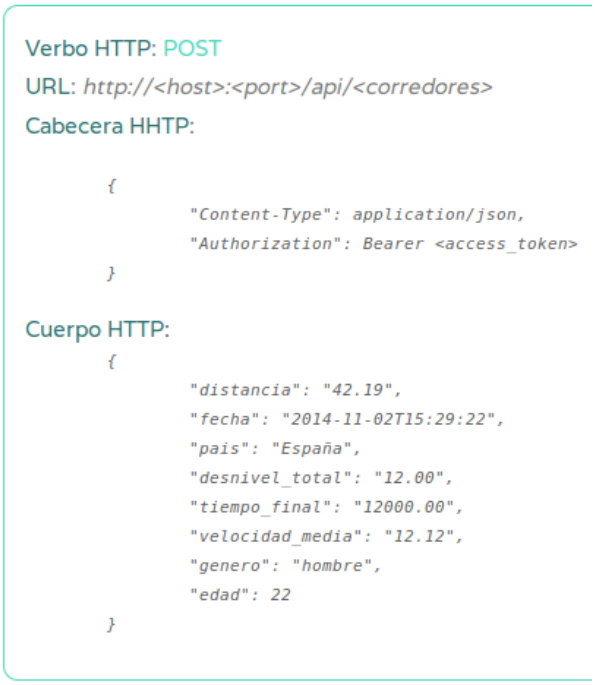


Figura Diseño-18: Ejemplo de petición para añadir un corredor

Como se puede observar en estas figuras, en la **cabecera** de la petición se incluye el formato del contenido del cuerpo de la misma, es decir, la forma en que se van a interpretar los datos, y el token de autenticación así como el protocolo, mientras que en el **cuerpo** se incluyen los atributos de la entidad en el formato especificado (p.e. JSON). La forma de conseguir el token de autenticación se explica en el Anexo B.

3.3.2 Leer objetos mediante la API

Para leer o recuperar un objeto concreto de la base de datos utilizaremos peticiones GET con la estructura mostrada en la figura 19. Como podemos ver, es necesario el identificador del objeto para recuperar su información.

Figura Diseño-19: Estructura general de una petición para leer un objeto

Por ejemplo, para leer un corredor con id = 56671623811b16cb24ebf1f6, la dirección URL sería:

http://<host>:<port>/api/corredores/56671623811b16cb24ebf1f6

Esta petición nos devuelve los datos requeridos en formato JSON para la entidad especificada.

3.3.3 Eliminar objetos mediante la API

Para leer un objeto concreto necesitamos el identificador del mismo.

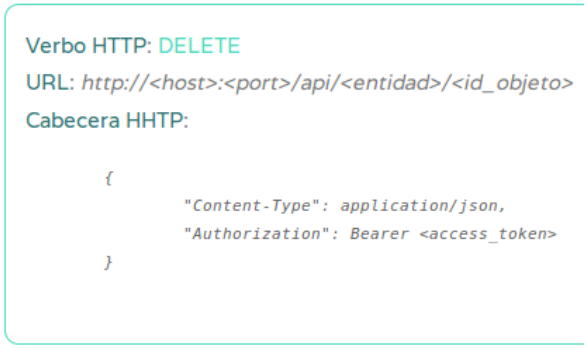


Figura Diseño-20: Estructura general de una petición para eliminar un objeto

Por ejemplo, para eliminar un corredor con id = 56671623811b16cb24ebf1f6, la dirección URL sería:

http://<host>:<port>/api/corredores/56671623811b16cb24ebf1f6

4 Desarrollo

En esta sección se desarrollará el proceso que se ha llevado a cabo paso por paso.
Se explicará cómo se han tratado los ficheros de datos, cómo se ha creado la base de datos, cómo se ha hecho la carga inicial de datos, el módulo REST del servidor y las funciones del modelo de aprendizaje automático.

4.1 Tratamiento de los datos

Para desarrollar este trabajo hemos recibido dos ficheros de datos por parte de la empresa RUNATOR que se dedica al desarrollo de aplicaciones informáticas en el sector del *running*. Por ello, nos han podido proporcionar datos de corredores anónimos junto con sus entrenamientos.

4.1.1 Ficheros de datos

El primer fichero corresponde a **corredores**, y tiene un total de 2914 ejemplos. Cada uno de ellos almacena información sobre:

- Id corredor: identifica de forma única a cada corredor, tiene formato UUID.
- Distancia: distancia de la maratón (42 km en todos los casos).
- Fecha: indica cuándo se produjo la carrera. Este campo será necesario para realizar el posterior procesamiento de los datos como veremos en la sección 4.1.2.
- Ciudad: ciudad de celebración de la maratón.
- País: país de celebración de la carrera.
- Desnivel Total: desnivel acumulado de la prueba.
- Tiempo Final: marca obtenida por el corredor en la prueba.
- Velocidad Media: velocidad mantenida por el corredor en promedio en minutos por kilómetro.
- Género: indica si el corredor es hombre o mujer.
- Edad: edad en años del corredor.

El otro fichero tiene datos sobre 6017 **entrenamientos**, realizados por los corredores del anterior fichero. Los registros de este fichero tienen los siguientes atributos:

- Id corredor: identificador del corredor, que hace referencia a un corredor del fichero anterior..
- Id entrenamiento: identificador del entrenamiento.
- App: aplicación desde la que se han obtenido los datos (Ver sección 2.5).
- Fecha: fecha del entrenamiento.
- Distancia total: distancia del entrenamiento.
- Tiempo total: duración del entrenamiento.
- Desnivel total: desnivel del entrenamiento.

A partir de este campo, el número de atributos es variable, y depende del número de intervalos en que esté divido el entrenamiento. En nuestro caso, se ha dividido el entrenamiento en intervalos de un kilómetro. Para cada uno de estos intervalos, encontramos los siguientes campos:

- Distancia intervalo: distancia del intervalo.
- Tiempo intervalo: tiempo del intervalo.
- Media intervalo: ritmo promedio seguido en el intervalo.

Para cargar estos datos en la base de datos, se han realizado varios *scripts* en Python. Gracias a Django ha sido posible interactuar con la base de datos de forma más sencilla mediante estos *scripts* de **carga inicial** de datos. Ha sido necesario procesar ciertos datos, por ejemplo, los tiempos, expresados en horas, minutos y segundos en los ficheros, se han convertido a segundos para que sea más fácil trabajar con ellos.

Además se han eliminado entrenamientos que, habiendo sido tomados por diferentes aplicaciones, estaban repetidos ya que comenzaban a la misma hora.

El resto de filtros se han aplicado posteriormente, tras haber cargado la base de datos (Ver sección 4.1.2).

4.1.2 Creación de histogramas

Una vez que hemos limpiado los datos y que los hemos cargado en la base de datos del servidor, hemos procedido a crear un *dataset* en un formato adecuado para los algoritmos de aprendizaje. Los algoritmos de aprendizaje necesitan que cada instancia de entrada a predecir venga dada por un vector de atributos. Dado que queremos predecir la marca de un corredor será necesario transformar todos los datos de sus entrenamientos, definidos en múltiples registros y tablas, a un único vector de atributos.

Hemos buscado una forma de caracterizar todos los entrenamientos de un corredor, incluyendo todos sus intervalos, en una única tupla de datos. Es necesario que cada ejemplo sea representado como un *array*. Para ello hemos recorrido los entrenamientos de cada corredor acumulando las distancias recorridas en sus respectivos intervalos en un histograma de ritmos de carrera.

Las franjas del histograma se definen mediante tres parámetros: un valor de ritmo *mínimo,* un valor de ritmo *máximo* y la anchura de los intervalos del histograma. En la figura 21 se puede ver cómo se han formado los histogramas.

Para calcular los histogramas que caracterizan los entrenamientos de cada corredor se han usado los siguientes parámetros:

- Anchura del intervalo: 30 seg/km
- Mínimo: 3:00 min/km
- Máximo: 7:30 min/km

La siguiente tupla representa un **entrenamiento** de un corredor. El proceso consiste en recorrer los **intervalos** sumando la distancia recorrida a cada franja de ritmo que corresponda. Aquellas distancias recorridas a un ritmo mayor que 3:00 min/km se incluirán en un intervalo para <3:00 min/km y aquellas distancias recorridas a un ritmo superior 7:30 min/km se incluiran en un intervalo para ritmos mayores de 7:30 min/km.

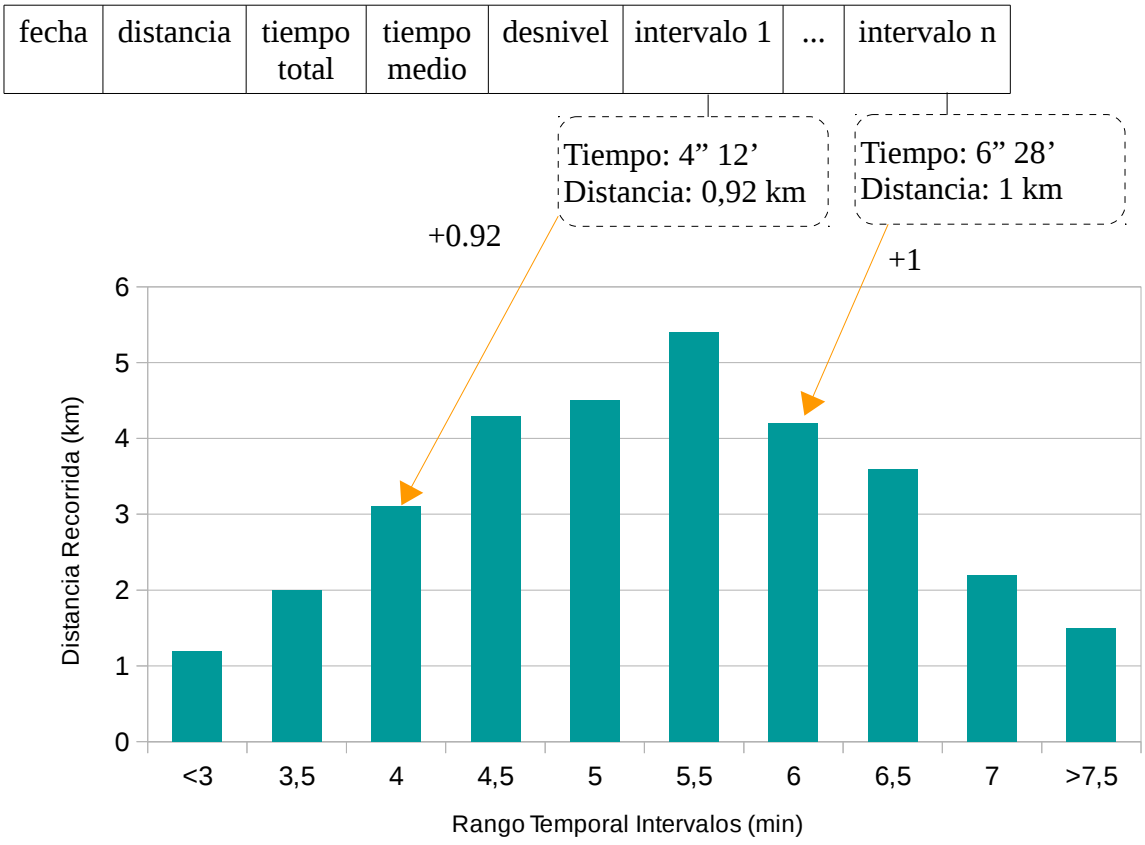


Figura Desarrollo-21: Histograma de un corredor

De esta forma, obtenemos un vector con el cúmulo de kilómetros recorridos por rango de velocidad. En nuestro ejemplo serían los valores de las barras verticales (en km):

[1.2, 2, 3.1, 4.3, 4.5, 5.4, 4.2, 3.6, 2.2, 1.5]

Pero no vamos a generar un único histograma, sino tres, que serán concatenados. El primero corresponde a los entrenamientos de los **dos meses anteriores** (a) a la prueba, el segundo al **mes -2** (b) con respecto a la maratón y el último al **mes anterior** (c) a la carrera.

Por último vamos a concatenar el **tiempo** que el corredor obtuvo en la carrera de referencia, como valor objetivo a predecir. Por tanto obtenemos un *array* con atributos-clase por cada corredor.

Histograma (a)	Histograma (b)	Histograma (c)	**tiempo final**

Figura Desarrollo-22: Atributos y clase del dataset final

Los entrenamientos que no correspondan a los dos meses anteriores no serán tenidos en cuenta, así como los corredores que no superen un cierto número de entrenamientos antes de la prueba. También se **filtran** las carreras demasiado lentas cuyo ritmo no es indicativo de una prueba constante (ritmos muy bajos, se considera que han ido andando).

Esta forma de plantear el problema no es la única posible, pero es útil en nuestro caso, con los datos de los que disponemos.

4.1.3 Particionado de los datos

Ya que tras el procesamiento de los datos y la creación de histogramas para obtener el *dataset* disponemos de un número bastante reducido de datos, es conveniente realizar una validación cruzada de forma que probemos todos los ejemplos como entrenamiento y como *test* al menos un vez. Hemos creado 10 particiones, utilizando la librería *scikit-learn* tal y como ya se ha descrito. Esto nos ha servido para validar los modelos que se van a crear en el servidor y tener una estimación de su precisión.

4.2 Predicción de marcas

Se han realizado pruebas con los algoritmos anteriormente explicados y se ha tratado el error obtenido con cada uno, tratando de minimizarlo. Los resultados se explican en profundidad en la sección de Resultados.

4.3 Servidor

Para ejecutar el servidor se ha decidido crear un entorno virtual. En él se han instalado las dependencias y módulos que se han usado en el proyecto, entre ellas Django.

Se ha generado un proyecto de Django, y se han importado los módulos para trabajar con el la arquitectura REST (Django REST Framework [8]).

Ha sido necesario incluir autenticación en el servidor para poder realizar acciones como la inserción de objetos en la base de datos. Esto se ha conseguido gracias al protocolo OAuth2. Para más detalles sobre su funcionamiento y sobre cómo autenticarse en el servidor, consultar el *Anexo B*.

5 Integración, pruebas y resultados

Vamos a repasar cuáles han sido los resultados obtenidos en este trabajo y cómo se podrían mejorar. Veremos qué pruebas se han realizado sobre cada módulo.

5.1 Probando la base de datos

Se han realizado varios *scripts* en Python para comprobar que es posible introducir y leer elementos de la base de datos. Mediante estos scripts se vuelcan todos los datos de los ficheros iniciales (en csv) en la base de datos.

Por otro lado, se ha comprobado que tanto desde el panel de administración de Django como desde el panel de la API REST también es posible añadir y leer objetos a nuestra base de datos.

Tras ejecutar los scripts para insertar los corredores en la base de datos, lo más importante es comprobar desde la API REST que los datos se han cargado correctamente mediante el método GET o accediendo a la URI del recurso en cuestión.
Todas estas pruebas han funcionado sin problema.

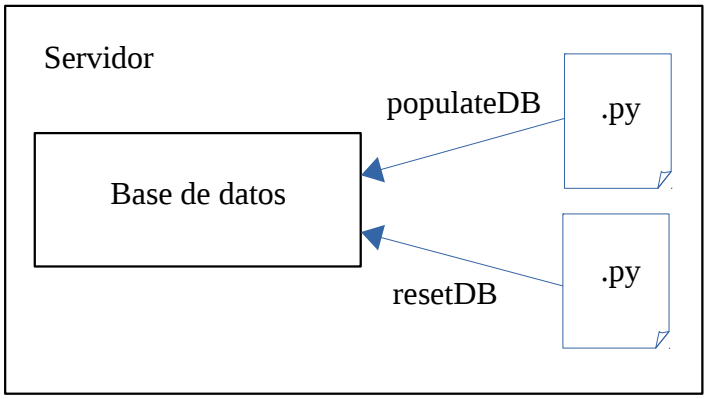


Figura Resultados-23: Pruebas locales con la base datos

5.2 Probando la API REST del servidor

Para comprobar que la API REST era accesible a través del protocolo HTTP se han realizado scripts en Python que interactúan con el servidor mediante dicho protocolo.

La lectura de los datos (GET) no dio ningún problema, no fue así la escritura. Para añadir un recurso mediante la API usamos una llamada POST, la cual daba problemas de identificación en un inicio, ya que no se habían tenido en cuenta cuestiones de seguridad. Este problema se solucionó utilizando el protocolo de seguridad OAuth2 al ser necesario crear un usuario con permisos en el servidor para poder escribir recursos (*Anexo B*).

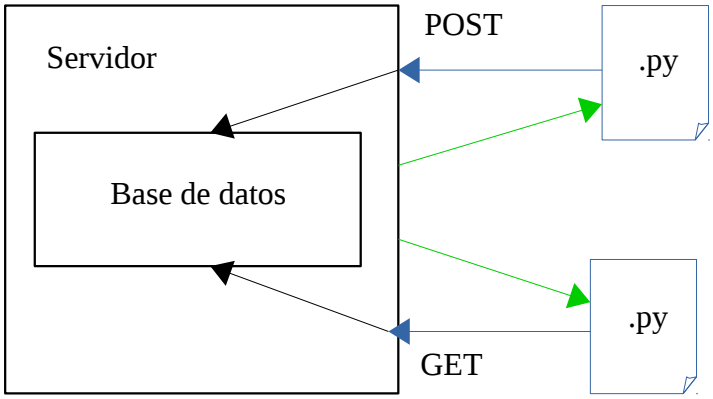


Figura Resultados-24: Pruebas remotas con la API

5.3 Resultados del Aprendizaje Automático

Probablemente sea la parte más importante del proyecto, ya que nos permite visualizar cómo han aprendido los modelos propuestos.

Es importante conocer cuánto nos estamos alejando del resultado esperado, es decir, qué error estamos obteniendo. Para ello es necesario analizar el problema. Ya que estamos usando regresores, y nuestras predicciones son valores reales, no podemos expresar nuestro error como un porcentaje sino que debemos utilizar un método que tenga en cuenta la distancia que separa nuestra predicción del valor de referencia. Por ello podemos utilizar métodos como el *error cuadrático medio* o como en nuestro caso el ***error absoluto medio (MAE)***, el cual se define como:

$$MAE(y,\hat{y}) = \frac{1}{n_{ejemplos}} \sum_{i=0}^{n_{ejemplos}-1} |y_i - \hat{y}_i|$$

donde medimos la **distancia** (valor absoluto) que nos separa del valor real en **promedio** por todos los ejemplos.
Para tratar de **minimizar** este error, hemos hecho pruebas con distintos parámetros y modelos. Además, hemos variado el **rango temporal** de los intervalos histogramas de los corredores. A continuación, podemos ver el error obtenido para distintos valores del rango temporal

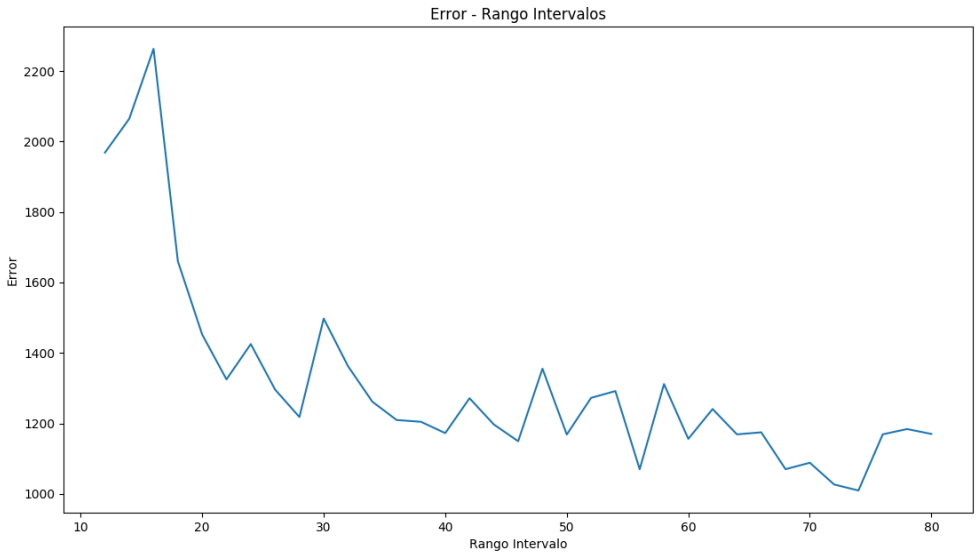


Figura Resultados-25: MAE frente al Rango temporal

El menor error obtenido tras varias pruebas se obtiene para un rango de aproximadamente 74 segundos, lo que es razonable ya que si el rango es muy bajo tendremos muchas franjas en el histograma y por tanto un gran número de ellas quedará vacío y si es muy alto, tendremos muy pocos atributos de entrada para nuestro modelo. En cualquiera de los casos se dan patrones poco representativos y demasiado comunes en todos los corredores. Por tanto el valor obtenido es adecuado y es el que usaremos para medir el error en los algoritmos siguientes.

Recordemos los algoritmos utilizados.

5.3.1 Linear Regression

Hemos representado los valores predichos frente a los valores reales para cada corredor, así como la recta que representa la predicción idónea para este modelo en la figura 26. En la figura, cada punto azul corresponde a un corredor.

El error absoluto medio obtenido es de aproximadamente 921 segundos, lo que equivale a aproximadamente 15 minutos.

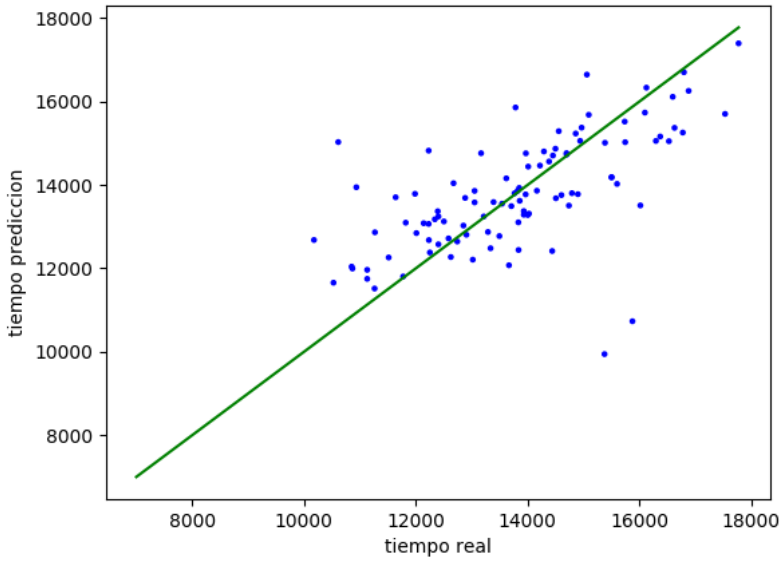


Figura Resultados-26: Resultados de Linear Regression

5.3.2 Random Forest Regressor

El resultados medio obtenido para la misma métrica del error para un *Random Forest* ha sido de 965 segundos que equivale a 16 minutos aproximadamente. En la figura se muestran las predicciones frente a las marcas reales para cada corredor, así como la predicción idónea. Se han definido un número total de 1000 estimadores (árboles del *Random Forest*) y el número de atributos para evaluar la mejor partición mediante la función *sqrt* (raíz cuadrada del número de atributos).

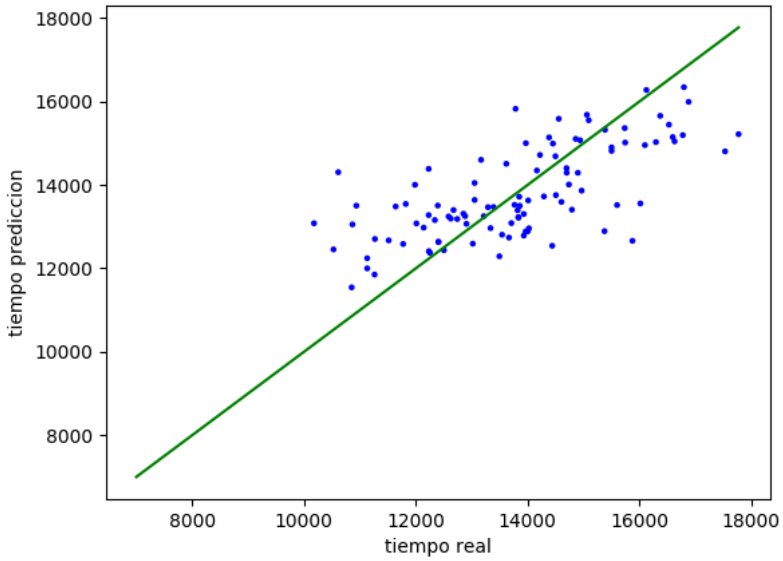


Figura Resultados-27: Resultados de Random Forest

5.3.3 Multilayer Perceptron Regressor

Los resultados obtenidos para la misma métrica del error han sido muy malos (superando los 1300 segundos) por lo que se ha descartado el uso de este modelo. Se han probado distinto número de capas ocultas y distinto número de neuronas en cada una. Si bien es cierto que probando más valores para el número de capas se podrían haber obtenido mejores resultados.

5.3.4 Análisis de los resultados

Conviene analizar los resultados obtenidos, ya que éstos no han sido tan buenos como hubieramos esperado. Teniendo en cuenta la duración tan elevada de una prueba de maratón, que para nuestros datos es de 13801.76 segundos ~ 3 horas y 50 minutos en promedio, no parece que un error de 15 minutos sea desorbitado. Sin embargo, teniendo en cuenta otros estudios llevados a cabo con anterioridad sobre este campo, no son resultados positivos. En algunas ocasiones se ha llegado a obtener un error de 13 minutos, aunque sigue siendo algo elevado.

Es interesante no quedarse en este punto, y preguntarse el porqué de este ligero aumento del error. El proyecto tiene una buena base y está bien planteado, de hecho se puede apreciar que los resultados se aproximan considerablemente a los valores reales. Entonces, ¿dónde podemos mejorar? ¿que ha fallado? ¿es un modelo realista?

Tras discutir los resultados, se ha llegado a la conclusión de que lo **datos** son **insuficientes**. Las razones que nos llevan a plantear esto son las siguientes. Para empezar, muchos de los corredores que estaban presentes en el fichero inicial no tenían asociados entrenamientos en el otro fichero (el de entrenamientos), por tanto no nos eran útiles. Un corredor que no ha entrenado o del cual no tenemos acceso a sus entrenamientos no aporta información al modelo de aprendizaje. El número de corredores válidos después de hacer la limpieza necesaria de los datos se ha quedado en torno a 101, dependiendo de los filtros que se apliquen (número de entrenamientos mínimo, etc). Los estudios previos habían dado resultados algo mejores pero usando en torno a 2000 corredores.

Por otro lado, los datos se han filtrado, por ejemplo, eliminando entrenamientos repetidos varias veces (tomados desde distintas *apps* al mismo tiempo) o descartando marcas demasiado altas en la carrera (se consideraba que el corredor había ido andando o a un ritmo demasiado bajo). Aún así, aparecen ciertas anomalías en los datos que no se han podido detectar y que casi con toda seguridad introducen ruido en la regresión.

Habría sido interesante además incluir otros atributos (como el género, la edad, el peso, la altura, etc) a los parámetros de entrada, además de los histogramas, pero no estaban presentes (o no existían) en los datos de todos los corredores.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Este trabajo es la unión de varios campos, por una lado el desarrollo de servidores y por otro el aprendizaje automático. Considero que se han consolidado conceptos aprendidos a lo largo del Grado en Ingeniería Informática, relacionados con los sistemas distribuidos y la inteligencia artificial.

Se ha desarrollado un servicio web que permite a un corredor predecir su marca en una maratón a partir de sus entrenamientos. Esto ha sido posible gracias a un framework web en Python como Django. Se han utilizado datos reales anónimos facilitados por la empresa RUNATOR, que se han tratado y almacenado en una base de datos de tipo SQLite3 dentro del servidor web.

Se ha implementado un servidor de tipo *RESTful*, al que se accede mediante una API REST, tanto a los recursos de la base de datos como al resto de acciones posibles del servicio (entrenar modelos de aprendizaje y predecir marcas de carreras).

Para realizar predicciones de marcas de maratón se han procesado los datos ya almacenados en la base de datos y se han utilizado en algoritmos de aprendizaje automático como *Regresión Lineal*, *Random Forest* y *Perceptrón Multicapa*.

Aunque los resultados no han sido tan buenos como se esperaba, sabemos que se debe a la escasez de datos de los que disponemos, pero los resultados muestran una tendencia favorable.

La base del proyecto no es compleja. La forma de resolver el problema que se nos plantea es adecuada ya que utiliza información interna de los entrenamientos de los corredores, por lo que los resultados se aproximan bastante a los obtenidos en las carreras reales, aunque existen otras formas alternativas de resolverlo.

La escasez de datos no supone un problema si se quiere continuar el trabajo en un futuro, ya que se pueden incluir mediante los métodos de inserción en la base de datos ya mencionados. Lo que pretendo transmitir es que aunque los resultados son mejorables, la forma de plantear el problema es adecuada y confío en que este proyecto sea de gran ayuda en un futuro.

6.2 Trabajo futuro

La empresa que se interesó por el proyecto y que nos facilitó los datos puede hacer uso de este trabajo como una versión inicial de una posible futura herramienta. Es importante que se añadan muchos más datos al proyecto así como adaptar el proyecto a sus necesidades y optimizar al máximo los modelos de aprendizaje para que fuese realmente útil, pero es una buena base de la que partir.

Referencias

1. *El auge del Running: http://economia.elpais.com/economia/2017/03/16/actualidad/1489678754_441758.html*

2. *Aplicaciones Smartwatches: https://es.wikipedia.org/wiki/Reloj_inteligente*

3. *Arquitectura RESTful: https://www.ibm.com/developerworks/ssa/library/wa-aj-multitier/*

4. *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*

5. *Métodos de predicción de marcas: http://www.runningforfitness.org/faq/vo2-max*

6. *Predicción en carreras de fondo: https://repositorio.uam.es/handle/10486/675353*

7. *Aprendizaje Automático: https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico*

8. *Regresión Lineal: https://es.wikipedia.org/wiki/Regresi%C3%B3n_lineal*

9. *Random Forest: http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics*

10. *Fundamentals of Neural Networks: Architectures, Algorithms, and Application, 1994, Laurene Fausett*

11. *Scikit Learn: http://scikit-learn.org/stable/*

12. *Matplotlib Python: https://matplotlib.org/*

13. *Runator About: https://www.runator.com/es/sobre-nosotros*

14. *Django Site: https://www.djangoproject.com/download/*

15. *Django REST Framework Official Documentation: http://www.django-rest-framework.org/*

16. *Django Installation: https://tutorial.djangogirls.org/es/django_installation/*

Glosario

API	Application Programming Interface
REST	Representational State Transfer
HTTP	Hypertext Tranfer Protocol
CSV	Comma Separated Values
URI	Uniform Resource Identifier
JSON	Javascript Oriented Notation
CRUD	Create, Read, Update, Delete
URL	Uniform Resource Locator
MAE	Mean Absolute Error

Anexos

A. Manual de instalación

Para este proyecto se ha implementado un servidor web basado en REST. Como framework se ha elegido *Django* [14], y adicionalmente se ha hecho uso de un módulo llamado *Django REST Framework* [15], que nos ha permitido desarrollar de forma más fácil el comportamiento del servicio ante las llamadas web.

Con el objetivo de aislar al máximo el servidor y evitar ciertas dependencias de python y otras librerías y paquetes, se ha instalado un **entorno virtual** dentro del proyecto, y es necesario ejecutarlo en este contexto. De esta forma se aislará la configuración de *Django/Python* en base a cada proyecto. Este entorno contiene herramientas como *Django* y *Django REST Framework* entre otras.

Veamos una pequeña introducción sobre la instalación de los requisitos del proyecto en el sistema operativo Ubuntu [16].

Para crear el entorno virtual primero debemos elegir en qué directorio vamos a trabajar.

```
mkdir my-env-dir
cd my-env-dir
```

El siguiente paso es crear nuestro entorno virtual:

```
python3 -m venv my-v-env
```

Se creará una carpeta que contiene los archivos del entorno virtual. Para trabajar en él debemos activarlo de la siguiente forma:

```
source my-v-env/bin/activate
```

O, de forma alternativa:

```
. my-v-env/bin/activate
```

A partir de este entonces estaremos dentro del entorno y observaremos que la ruta del sistema aparece de la siguiente forma:

```
(my-v-env) ~/my-env-dir$
```

A continuación deberíamos instalar ***Django*** en su última versión, la 1.10.6:

```
pip install Django==1.10.6
```

aunque no será necesario en nuestro caso, ya que está instalado y configurado.

B. Manual del programador

Una vez instalado y preparado el entorno, podemos realizar una serie de acciones en el servidor:

Principalmente neceitaremos

```
python manage.py runserver
```

Acceder al servidor de forma remota

Para acceder al servidor mediante la API REST se utiliza el protocolo OAuth2. Es necesario obtener un token de autenticación, que se le otorgará cuando realice un login correcto en el servidor, y que deberá guardar para las posteriores llamadas al servidor. Para ello es necesario primero crear un usuario. Elegiremos *nombre* y *contraseña*:

```
python manage.py createsuperuser
```

Además debemos crear una aplicación en el servidor, desde la siguiente dirección:

```
http://<host>:<port>/o/applications/
```

La debemos configurar con "Client type" a "Confidential" y "Authorization grant type" a "Resource owner password-based". La dirección URL para acceder es la siguiente, mediante el método POST:

```
http://<host>:<port>/o/token/?
grant_type=password&username=<nombre>&password=<contraseña>&
client_id=<client_id>&client_secret=<client_secret>
```

Una vez nos hayamos identificado, el servidor nos contestará con el **token** de autenticación y lo debemos adjuntar en la cabecera de todas las peticiones HTTP que enviemos, de la siguiente forma:

```
Authorization: Bearer <access_token>
```

Envío de peticiones autenticadas

Vamos a ver un ejemplo del envío de una petición al servidor. Este caso, tras crear un usuario, nos identificamos y pedimos el *access token*.

```
def login(username, password):
user_data = {
	'username': username,
	'password': password
}
login_url = 'http://localhost:8000/o/token/?
grant_type=password&username=admin&password=admin@admin&client_id=rIwywpAnTucPewjV1S6g8wf
kwl3YhHecJ7knHvlP&client_secret=LsAUF1Lr01yNA0RZTElgSTWat6k9a8ORnV1VhueOwtXdp9TAjMQgwgOQb
cy2driBOjOp4zMkfMV0jUqU41lX5oQoFtNLVTKUYItbTYqL067MRPIVv1hZBPvClAfse3Ur'
headers = {
	'content-type': 'application/json'
}
r = requests.post(login_url, data=json.dumps(user_data), headers=headers)
content = json.loads(r.content.decode('utf-8'))
access_token = content['access_token']
return access_token
```

Formamos la cabecera (*headers*), y el cuerpo (*body*). Después enviamos la petición POST para añadir por ejemplo un corredor.

```
{
"fecha": "2016-05-02",
"lugar": "Madrid",
"desnivel_total": 7,
"tiempo_media": 200,
"tiempo_final": 450
}
```

```
{
Authentication: Bearer <access_token>,
Content-Type: application/json
}
```